# MICROPROCESADOR (CÓDIGO EN VHDL) TOMADO DEL LIBRO PARDO Y BOLUDA

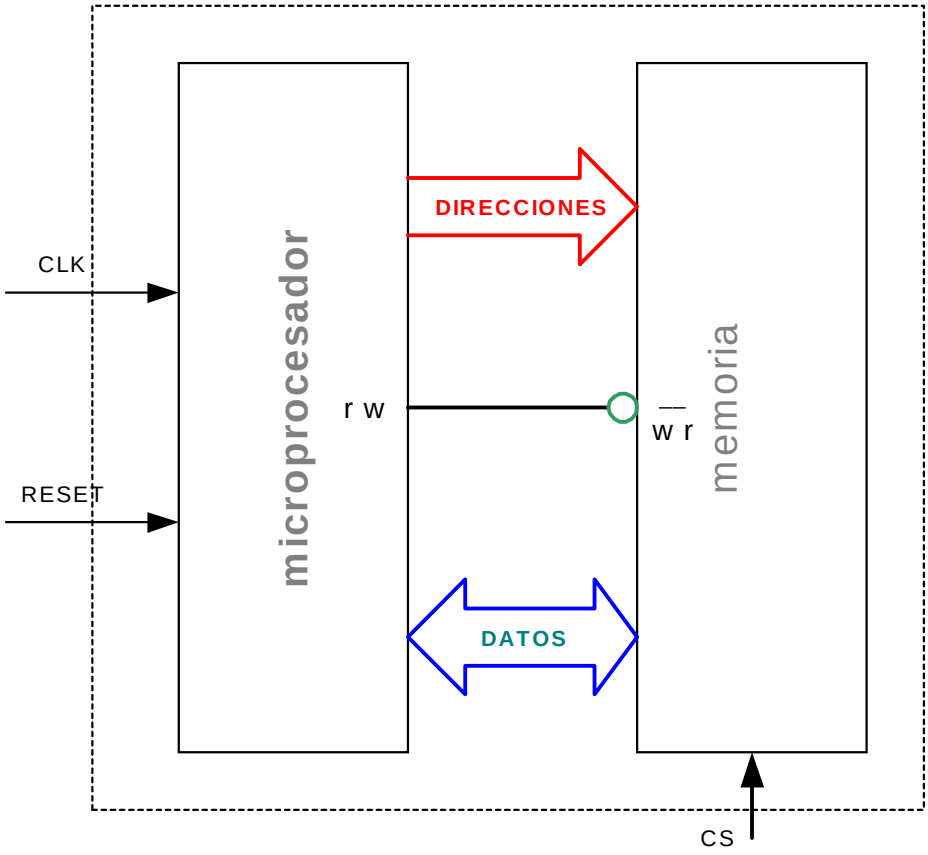
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY procesador IS
PORT(clk,rst: IN std_logic;
     r_w: OUT std_logic;
     dir: OUT std_logic_vector(7 DOWNTO 0);
     dat: INOUT std_logic_vector(7 DOWNTO 0));
END procesador;

ARCHITECTURE descripcion OF procesador IS
  TYPE estado IS
(inicial,busqueda,ejec,ldxxa,ldaxx,anda,adda,suba);
  SIGNAL a,pc,carga_a,carga_pc: std_logic_vector(7 DOWNTO 0);
  SIGNAL rdat_in,dat_in,dat_out: std_logic_vector(7 DOWNTO 0);
  SIGNAL rwaux: std_logic;
  SIGNAL presente: estado:=inicial;
--inicio de la funcion +
  FUNCTION "+"(d1,d2: std_logic_vector(7 DOWNTO 0)) RETURN
    std_logic_vector IS
  VARIABLE sal: std_logic_vector(7 DOWNTO 0);
  VARIABLE carry: std_logic:='0';
  VARIABLE ind: INTEGER;
  BEGIN
     FOR ind IN 0 TO 7 LOOP
         sal(ind):=d1(ind) XOR d2(ind) XOR carry;
         carry:=(d1(ind) AND d2(ind)) OR (d1(ind) AND carry) OR
(d2(ind) AND carry);
     END LOOP;
     RETURN sal;
  END "+";
--fin de la funcion +

--inicio de la funcion -
     FUNCTION "-"(d1,d2: std_logic_vector(7 DOWNTO 0)) RETURN
    std_logic_vector IS
  BEGIN
     RETURN d1+((NOT d2)+"00000001");
  END "-";
--fin de la funcion -

BEGIN

fsm:
PROCESS(clk,rst)
BEGIN
 IF rst='1' THEN
```

```
    presente<=inicial;
 ELSIF clk='1' AND clk'event THEN
  CASE presente IS
  WHEN inicial =>
     presente<=busqueda;
  WHEN busqueda=>
     IF dat_in(2 DOWNTO 0)="111" THEN presente<=busqueda;
     ELSE presente<=ejec;
     END IF;
  WHEN ejec =>
     CASE rdat_in(2 DOWNTO 0) IS
     WHEN "000" =>
        presente<=ldaxx;
     WHEN "001" =>
        presente<=ldxxa;
     WHEN "010" =>
        presente<=anda;
     WHEN "011" =>
        presente<=adda;
     WHEN "100" =>
        presente<=suba;
     WHEN OTHERS =>      -- jz,jmp
        presente<=busqueda;
     END CASE;
  WHEN OTHERS=>
     presente<=busqueda;
  END CASE;
 END IF;
END PROCESS fsm;

salida:
PROCESS(presente,pc,a,rdat_in,dat_in)
BEGIN
  CASE presente IS
  WHEN inicial =>
    carga_pc<="00000000";     -- PC a 0
    carga_a<="00000000";      -- Acumulador a 0
    dir<=pc;                  -- Dirección a 0;
    rwaux<='1';               -- Lectura
  WHEN busqueda=>
    carga_pc<=pc+"00000001";  -- Incrementa PC
    carga_a<=a;               -- Deja A como estaba
    dir<=pc;
    rwaux<='1';
  WHEN ejec =>
    CASE rdat_in(2 DOWNTO 0) IS
    WHEN "101" =>  -- jz
       dir<=pc;
       IF a="00000000" THEN
         carga_pc<=dat_in;   -- Salta si A=0
       END IF;
       rwaux<='1';
       carga_a<=a;
    WHEN "110" =>  --jmp
       dir<=pc;
       carga_pc<=dat_in;     -- Salta siempre
```

```
        rwaux<='1';
        carga_a<=a;
      WHEN OTHERS =>   -- Por defecto para el resto:
        rwaux<='1';   -- Leer
        dir<=pc;      -- Dirección del operando
        carga_pc<=pc;
        carga_a<=a;
      END CASE;
    WHEN ldaxx =>
      rwaux<='1';
      carga_a<=dat_in;
      carga_pc<=pc+"00000001";
      dir<=rdat_in;
    WHEN ldxxa =>
      rwaux<='0';   -- Único que escribe
      carga_a<=a;
      carga_pc<=pc+"00000001";
      dir<=rdat_in;
    WHEN anda =>
      rwaux<='1';
      carga_a<=a AND dat_in;
      carga_pc<=pc+"00000001";
      dir<=rdat_in;
    WHEN adda =>
      rwaux<='1';
      carga_a<=a+dat_in;
      carga_pc<=pc+"00000001";
      dir<=rdat_in;
    WHEN suba =>
      rwaux<='1';
      carga_a<=a-dat_in;
      carga_pc<=pc+"00000001";
      dir<=rdat_in;
    END CASE;
  END PROCESS salida;

  registro_de_entrada:
  PROCESS(clk,rst)
  BEGIN
    IF rst='1' THEN rdat_in<="00000000";
    ELSIF clk='1' AND clk'event THEN rdat_in<=dat_in;
    END IF;
  END PROCESS registro_de_entrada;

  regs_con_carga:
  PROCESS(clk,rst)
  BEGIN
    IF rst='1' THEN
      a <="00000000";
      pc<="00000000";
    ELSIF clk='1' AND clk'event THEN
      a <=carga_a;
      pc<=carga_pc;
    END IF;
  END PROCESS regs_con_carga;
```

```
r_w<=rwaux;   -- Para leer la salida.

-- Triestado de salida:
dat_in<=dat;
dat_out<=a;
dat<=dat_out WHEN rwaux='0' ELSE "ZZZZZZZZ";

END descripcion;
```

2.- Para la segunda parte se pide que hagamos un mapeo de memoria, para lo cual yo creo una entidad global (algo así como un microcontrolador), donde yo incluyo internamente los componentes memoria y procesador, y genero sólo tres señales de entrada CLK; RESET y CHIP SELECT (CS) de la memoria), y saco las lineas de datos, direcciones y el acumulador solamente para monitoreo.




Para esto grabamos en la memoria los siguientes datos:

| DIRECCIÓN | DATO | INSTRUCCIÓN |
|---|---|---|
| F0H | 30H | LD A, 31H |
| F1H | 31H | -- |
| F2H | 32H | AND A,33H |
| F3H | 33H | -- |
| F4H | 34H | SUB A, 00H (por defecto) |

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity memoria_datos is
    Port ( direcciones : in std_logic_vector(7 downto 0);
           datos : out std_logic_vector(7 downto 0);
           wr : in std_logic;
           cs : in std_logic);
end memoria_datos;
architecture Behavioral of memoria_datos is

begin
parte1:    process(direcciones,wr,cs)
                begin
                if cs='1' then
                     if wr='1' then
                          case direcciones is
                          when "11110000" => datos<="00110000";
                          when "11110001" => datos<=
     "00110001";
                          when "11110010" => datos<=
     "00110010";
                          when "11110011" => datos<=
     "00110011";
                          when "11110100" => datos<=
     "00110100";
                          when others => datos <="00000000";
                          end case;
                     end if;
             end if;
             end process;
end Behavioral;
```

Y EL ARCHIVO SISTEMA SERÁ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sistema_micro_memoria is
    Port ( clk : in std_logic;
           reset : in std_logic;
           cs : in std_logic;
                  wr1: out std_logic;
                  dir_final:out std_logic_vector (7 downto 0);
                  datos_final:out std_logic_vector (7 downto
0));
end sistema_micro_memoria;

architecture Behavioral of sistema_micro_memoria is
signal wr: std_logic:='1';
signal di,da: std_logic_vector (7 downto 0);
component procesador
PORT(clk,rst: IN std_logic;
```

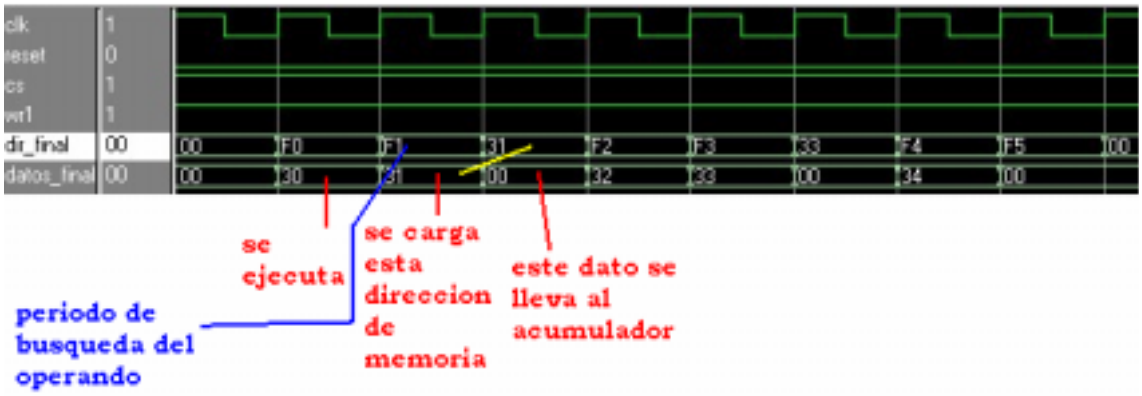
```
     r_w: OUT std_logic;
     dir: OUT std_logic_vector(7 DOWNTO 0);
     dat: INOUT std_logic_vector(7 DOWNTO 0));
END component;
component memoria_datos
    Port ( direcciones : in std_logic_vector(7 downto 0);
           datos : out std_logic_vector(7 downto 0);
           wr : in std_logic;
           cs : in std_logic);
END component;

begin

u0: memoria_datos port map(di,da,wr,cs);
u1: procesador port map (clk,reset,wr,di,da);
wr1<=wr;
dir_final<=di;
datos_final<=da;
end Behavioral;
```

Y LUEGO CREAMOS EL ARCHIVO DE SIMULACIÓN

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

     COMPONENT sistema_micro_memoria
     PORT(
          clk : IN std_logic;
          reset : IN std_logic;
          cs : IN std_logic;
          wr1 : OUT std_logic
          );
     END COMPONENT;

     SIGNAL clk :  std_logic:='1';
     SIGNAL reset :  std_logic:='1';
     SIGNAL cs :  std_logic;
     SIGNAL wr1 :  std_logic;

BEGIN

     uut: sistema_micro_memoria PORT MAP(
          clk => clk,
          reset => reset,
          cs => cs,
          wr1 => wr1
     );
clk<=not clk after 50 ns;
reset<= '0' after 50 ns;
```

```
cs<='1';

END;
```

Y OBTENEMOS LA SIGUIENTE GRÁFICA




3.- AHORA LE VAMOS A AUMENTAR ALGUNAS INSTRUCCIONES MAS:

| HEXADECIMAL | INSTRUCCIÓN | BINARIO |
|---|---|---|
| 00H | LD A,xx | 00000000 |
| 01H | LD xx,A | 00000001 |
| 02H | AND A,xx | 00000010 |
| 03H | ADD A,xx | 00000011 |
| 04H | SUB A,xx | 00000100 |
| 05H | JZ xx | 00000101 |
| 06H | JMP xx | 00000110 |
| 07H | NOP | 00000111 |
| 09H | OR A,xx | 00001000 |
| 0AH | XOR A,xx | 00001001 |
| 0BH | NOT A | 00001010 |
| 0CH | INC A | 00001011 |
| 0DH | DEC A | 00001100 |
| 0EH | --- | 00001110 |
| 0FH | --- | 00001111 |

IMPLEMENTAMOS EL SIGUIENTE CODIGO PARA EL MICROPROCESADOR:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY procesador_v2 IS
PORT(clk,rst: IN std_logic;
     r_w: OUT std_logic;
     dir: OUT std_logic_vector(7 DOWNTO 0);
     dat: INOUT std_logic_vector(7 DOWNTO 0));
END procesador_v2;

ARCHITECTURE descripcion OF procesador_v2 IS
```

```
  TYPE estado IS
(inicial,busqueda,ejec,ldxxa,ldaxx,anda,adda,suba,oraxx,xoraxx,n
ota,inca,deca);
  SIGNAL a,pc,carga_a,carga_pc: std_logic_vector(7 DOWNTO 0);
  SIGNAL rdat_in,dat_in,dat_out: std_logic_vector(7 DOWNTO 0);
  SIGNAL rwaux: std_logic;
  SIGNAL presente: estado:=inicial;
--inicio de la funcion +
  FUNCTION "+"(d1,d2: std_logic_vector(7 DOWNTO 0)) RETURN
    std_logic_vector IS
  VARIABLE sal: std_logic_vector(7 DOWNTO 0);
  VARIABLE carry: std_logic:='0';
  VARIABLE ind: INTEGER;
  BEGIN
     FOR ind IN 0 TO 7 LOOP
         sal(ind):=d1(ind) XOR d2(ind) XOR carry;
         carry:=(d1(ind) AND d2(ind)) OR (d1(ind) AND carry) OR
(d2(ind) AND carry);
     END LOOP;
     RETURN sal;
  END "+";
--fin de la funcion +

--inicio de la funcion -
      FUNCTION "-"(d1,d2: std_logic_vector(7 DOWNTO 0)) RETURN
    std_logic_vector IS
  BEGIN
     RETURN d1+((NOT d2)+"00000001");
  END "-";
--fin de la funcion -

BEGIN

fsm:
PROCESS(clk,rst)
BEGIN
 IF rst='1' THEN
    presente<=inicial;
 ELSIF clk='1' AND clk'event THEN
  CASE presente IS
  WHEN inicial =>
     presente<=busqueda;
  WHEN busqueda=>
     IF dat_in(3 DOWNTO 0)="0111" THEN presente<=busqueda;
     ELSE presente<=ejec;
     END IF;
  WHEN ejec =>
     CASE rdat_in(3 DOWNTO 0) IS
     WHEN "0000" =>
        presente<=ldaxx;
     WHEN "0001" =>
        presente<=ldxxa;
     WHEN "0010" =>
        presente<=anda;
     WHEN "0011" =>
        presente<=adda;
```

```
     WHEN "0100" =>
        presente<=suba;
     WHEN "1000" =>
        presente<=oraxx;
     WHEN "1001" =>
        presente<=xoraxx;
     WHEN "1010" =>
        presente<=nota;
     WHEN "1011" =>
        presente<=inca;
     WHEN "1100" =>
        presente<=deca;
     WHEN OTHERS =>      -- jz,jmp
        presente<=busqueda;
     END CASE;
  WHEN OTHERS=>
     presente<=busqueda;
  END CASE;
 END IF;
END PROCESS fsm;

salida:
PROCESS(presente,pc,a,rdat_in,dat_in)
BEGIN
  CASE presente IS
  WHEN inicial =>
    carga_pc<="00000000";     -- PC a 0
    carga_a<="00000000";      -- Acumulador a 0
    dir<=pc;                  -- Dirección a 0;
    rwaux<='1';               -- Lectura
  WHEN busqueda=>
    carga_pc<=pc+"00000001";  -- Incrementa PC
    carga_a<=a;               -- Deja A como estaba
    dir<=pc;
    rwaux<='1';
  WHEN ejec =>
    CASE rdat_in(3 DOWNTO 0) IS
    WHEN "0101" =>  -- jz
       dir<=pc;
       IF a="00000000" THEN
         carga_pc<=dat_in;   -- Salta si A=0
       END IF;
       rwaux<='1';
       carga_a<=a;
    WHEN "0110" =>  --jmp
       dir<=pc;
       carga_pc<=dat_in;     -- Salta siempre
       rwaux<='1';
       carga_a<=a;
    WHEN OTHERS =>   -- Por defecto para el resto:
       rwaux<='1';   -- Leer
       dir<=pc;      -- Dirección del operando
       carga_pc<=pc;
       carga_a<=a;
    END CASE;
  WHEN ldaxx =>
```

```
    rwaux<='1';
    carga_a<=dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN ldxxa =>
    rwaux<='0';   -- Único que escribe
    carga_a<=a;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN anda =>
    rwaux<='1';
    carga_a<=a AND dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN adda =>
    rwaux<='1';
    carga_a<=a+dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN suba =>
    rwaux<='1';
    carga_a<=a-dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN oraxx =>
    rwaux<='1';
    carga_a<=a or dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN xoraxx =>
    rwaux<='1';
    carga_a<=a xor dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN nota =>
    rwaux<='1';
    carga_a<=not a;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN inca =>
    rwaux<='1';
    carga_a<=a+"00000001";
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN deca =>
    rwaux<='1';
    carga_a<=a-"00000001";
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  END CASE;
END PROCESS salida;

registro_de_entrada:
PROCESS(clk,rst)
BEGIN
  IF rst='1' THEN rdat_in<="00000000";
```

```
  ELSIF clk='1' AND clk'event THEN rdat_in<=dat_in;
  END IF;
END PROCESS registro_de_entrada;

regs_con_carga:
PROCESS(clk,rst)
BEGIN
  IF rst='1' THEN
    a <="00000000";
    pc<="00000000";
  ELSIF clk='1' AND clk'event THEN
    a <=carga_a;
    pc<=carga_pc;
  END IF;
END PROCESS regs_con_carga;

r_w<=rwaux;   -- Para leer la salida.

-- Triestado de salida:
dat_in<=dat;
dat_out<=a;
dat<=dat_out WHEN rwaux='0' ELSE "ZZZZZZZZ";

END descripcion;
```

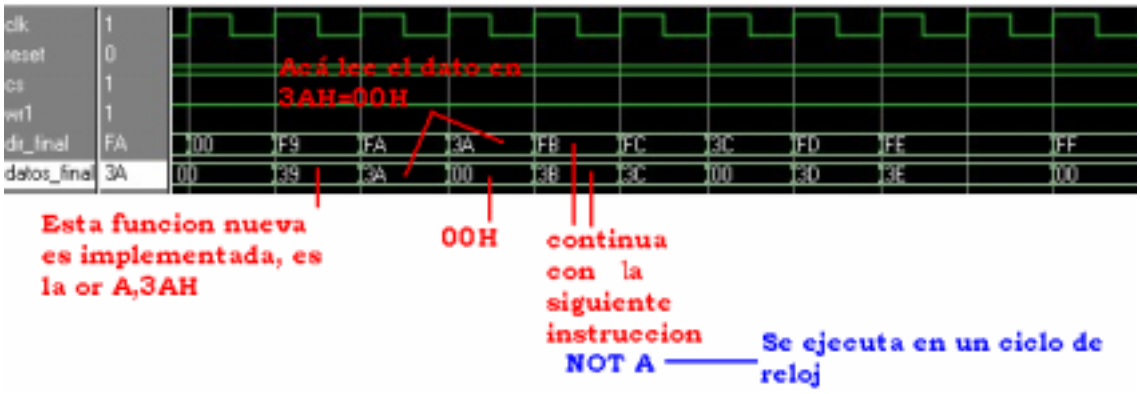
Y EL SIGUIENTE CODIGO DE MEMORIA PARA PROBARLO

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity memoria_datosv2 is
    Port ( direcciones : in std_logic_vector(7 downto 0);
           datos : out std_logic_vector(7 downto 0);
           wr : in std_logic;
           cs : in std_logic);
end memoria_datosv2;
architecture Behavioral of memoria_datosv2 is

begin
parte1:    process(direcciones,wr,cs)
                begin
                if cs='1' then
                     if wr='1' then
                          case direcciones is
                          when "11111001" =>datos<= "00111001";
                          when "11111010" =>datos<= "00111010";
                          when "11111011" =>datos<= "00111011";
                          when "11111100" =>datos<= "00111100";
                          when "11111101" =>datos<= "00111101";
                          when "11111110" =>datos<= "00111110";
                          when others => datos <="00000000";
                          end case;
                     end if;
```

```
            end if;
            end process;
end Behavioral;
```

**Y LUEGO EL ARCHIVO DEL SISTEMA SERÁ IGUAL PARA TODOS LOS CASOS, AL IGUAL QUE EL DE PRUEBA.**

OBTENEMOS LA SIGUIENTE IMAGEN




4.- Se pide incrementar dos registros mas, R1 y R2, los cuales sólo operarán a través de A, nos dan 4 instrucciones mas a implementar

| HEXADECIMAL | INSTRUCCIÓN | BINARIO |
|---|---|---|
| 00H | LD A,xx | 00000000 |
| 01H | LD xx,A | 00000001 |
| 02H | AND A,xx | 00000010 |
| 03H | ADD A,xx | 00000011 |
| 04H | SUB A,xx | 00000100 |
| 05H | JZ xx | 00000101 |
| 06H | JMP xx | 00000110 |
| 07H | NOP | 00000111 |
| 09H | OR A,xx | 00001000 |
| 0AH | XOR A,xx | 00001001 |
| 0BH | NOT A | 00001010 |
| 0CH | INC A | 00001011 |
| 0DH | DEC A | 00001100 |
| 0EH | MOV A,R1 | 00001110 |
| 0FH | MOV R1,A | 00001111 |
| 10H | MOV A,R2 | 00010000 |
| 11H | MOV R2,A | 00010001 |

Creamos el nuevo archivo para el procesador

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY procesador_version3 IS
PORT(clk,rst: IN std_logic;
     r_w: OUT std_logic;
```

```
     dir: OUT std_logic_vector(7 DOWNTO 0);
--Agrego estas tres seales para poder ver los registros en
tiempo real
       ta: out std_logic_vector(7 downto 0);
       tR1: out std_logic_vector(7 downto 0);
       tR2: out std_logic_vector(7 downto 0);
     dat: INOUT std_logic_vector(7 DOWNTO 0));
END procesador_version3;

ARCHITECTURE descripcion OF procesador_version3 IS
  TYPE estado IS
(inicial,busqueda,ejec,ldxxa,ldaxx,anda,adda,suba,oraxx,xoraxx);
  SIGNAL a,pc,carga_a,carga_pc: std_logic_vector(7 DOWNTO 0);
  SIGNAL R1,R2,CARGA_R1,CARGA_R2: std_logic_vector (7 downto 0);
  SIGNAL rdat_in,dat_in,dat_out: std_logic_vector(7 DOWNTO 0);
  SIGNAL rwaux: std_logic;
  SIGNAL presente: estado:=inicial;
--inicio de la funcion +
  FUNCTION "+"(d1,d2: std_logic_vector(7 DOWNTO 0)) RETURN
    std_logic_vector IS
  VARIABLE sal: std_logic_vector(7 DOWNTO 0);
  VARIABLE carry: std_logic:='0';
  VARIABLE ind: INTEGER;
  BEGIN
     FOR ind IN 0 TO 7 LOOP
         sal(ind):=d1(ind) XOR d2(ind) XOR carry;
         carry:=(d1(ind) AND d2(ind)) OR (d1(ind) AND carry) OR
(d2(ind) AND carry);
     END LOOP;
     RETURN sal;
  END "+";
--fin de la funcion +

--inicio de la funcion -
      FUNCTION "-"(d1,d2: std_logic_vector(7 DOWNTO 0)) RETURN
    std_logic_vector IS
  BEGIN
     RETURN d1+((NOT d2)+"00000001");
  END "-";
--fin de la funcion -

BEGIN

fsm:
PROCESS(clk,rst)
BEGIN
 IF rst='1' THEN
    presente<=inicial;
 ELSIF clk='1' AND clk'event THEN
  CASE presente IS
  WHEN inicial =>
     presente<=busqueda;
  WHEN busqueda=>
     IF dat_in(4 DOWNTO 0)="00111" THEN presente<=busqueda;
     ELSE presente<=ejec;
     END IF;
```

```
  WHEN ejec =>
     CASE rdat_in(4 DOWNTO 0) IS
     WHEN "00000" =>
        presente<=ldaxx;
     WHEN "00001" =>
        presente<=ldxxa;
     WHEN "00010" =>
        presente<=anda;
     WHEN "00011" =>
        presente<=adda;
     WHEN "00100" =>
        presente<=suba;
     WHEN "01000" =>
        presente<=oraxx;
     WHEN "01001" =>
        presente<=xoraxx;

     WHEN OTHERS =>      -- jz,jmp
        presente<=busqueda;
     END CASE;
  WHEN OTHERS=>
     presente<=busqueda;
  END CASE;
 END IF;
END PROCESS fsm;

salida:
PROCESS(presente,pc,a,rdat_in,dat_in)
BEGIN
  CASE presente IS
  WHEN inicial =>
    carga_pc<="00000000";     -- PC a 0
    carga_a<="00000000";      -- Acumulador a 0
    carga_R1<="00000000";      -- Acumulador R1 0
    carga_R2<="00000000";      -- Acumulador R2 0
    dir<=pc;                  -- Dirección a 0;
    rwaux<='1';               -- Lectura
  WHEN busqueda=>
    carga_pc<=pc+"00000001";  -- Incrementa PC
    carga_a<=a;               -- Deja A como estaba
    carga_R1<=R1;               -- Deja R1omo estaba
    carga_R2<=R2;               -- Deja R2omo estaba
    dir<=pc;
    rwaux<='1';
  WHEN ejec =>
    CASE rdat_in(4 DOWNTO 0) IS
    WHEN "00101" =>  -- jz
       dir<=pc;
       IF a="00000000" THEN
         carga_pc<=dat_in;   -- Salta si A=0
       END IF;
       rwaux<='1';
       carga_a<=a;
    WHEN "00110" =>  --jmp
       dir<=pc;
       carga_pc<=dat_in;     -- Salta siempre
```

```
      rwaux<='1';
      carga_a<=a;
    WHEN "01010" =>
          dir<=pc;
      rwaux<='1';
      carga_pc<=pc+"00000001";
          carga_a<=not a;
    WHEN "01011" =>
          dir<=pc;
      rwaux<='1';
      carga_pc<=pc+"00000001";
          carga_a<=a+"00000001";
    WHEN "01100" =>
          dir<=pc;
      rwaux<='1';
      carga_pc<=pc+"00000001";
          carga_a<=a-"00000001";
    WHEN "01101" =>
          dir<=pc;
      rwaux<='1';
      carga_pc<=pc+"00000001";
          carga_a<=R1;
    WHEN "01110" =>
          dir<=pc;
      rwaux<='1';
      carga_pc<=pc+"00000001";
          carga_R1<=a;
    WHEN "01111" =>
          dir<=pc;
      rwaux<='1';
      carga_pc<=pc+"00000001";
          carga_a<=R2;
    WHEN "10000" =>
          dir<=pc;
      rwaux<='1';
      carga_pc<=pc+"00000001";
          carga_R2<=a;
   WHEN OTHERS =>   -- Por defecto para el resto:
      rwaux<='1';   -- Leer
      dir<=pc;      -- Dirección del operando
      carga_pc<=pc;
      carga_a<=a;
          carga_R1<=R1;
          carga_R2<=R2;
   END CASE;
 WHEN ldaxx =>
   rwaux<='1';
   carga_a<=dat_in;
   carga_pc<=pc+"00000001";
   dir<=rdat_in;
 WHEN ldxxa =>
   rwaux<='0';   -- Único que escribe
   carga_a<=a;
   carga_pc<=pc+"00000001";
   dir<=rdat_in;
 WHEN anda =>
```

```
    rwaux<='1';
    carga_a<=a AND dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN adda =>
    rwaux<='1';
    carga_a<=a+dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN suba =>
    rwaux<='1';
    carga_a<=a-dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN oraxx =>
    rwaux<='1';
    carga_a<=a or dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN xoraxx =>
    rwaux<='1';
    carga_a<=a xor dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;

  END CASE;
END PROCESS salida;

registro_de_entrada:
PROCESS(clk,rst)
BEGIN
  IF rst='1' THEN rdat_in<="00000000";
  ELSIF clk='1' AND clk'event THEN rdat_in<=dat_in;
  END IF;
END PROCESS registro_de_entrada;

regs_con_carga:
PROCESS(clk,rst)
BEGIN
  IF rst='1' THEN
    a <="00000000";
      R1<="00000000";
      R2<="00000000";
    pc<="00000000";
  ELSIF clk='1' AND clk'event THEN
    a <=carga_a;
      R1<=carga_R1;
      R2<=carga_R2;
    pc<=carga_pc;
  END IF;
END PROCESS regs_con_carga;

r_w<=rwaux;   -- Para leer la salida.

-- Triestado de salida:
dat_in<=dat;
```

```
dat_out<=a;
dat<=dat_out WHEN rwaux='0' ELSE "ZZZZZZZZ";
--Zoma aumentada para pruebas
ta<=a;
tR1<=R1;
tR2<=R2;

END descripcion;
```

LO UNIMOS DEL MISMO MODO ANTERIOR EN UN SOLO ARCHIVO DE SISTEMA, CON LA SIGUIENTE ENTIDAD DE MEMORIA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sistema_procesador_version3_memoria is
   Port ( clk : in std_logic;
          reset : in std_logic;
          cs : in std_logic;
                        wr1: out std_logic;
                        dir_final:out std_logic_vector (7 downto 0);
                ta_final: out std_logic_vector(7 downto 0);
                        tR1_final: out std_logic_vector(7 downto 0);
                        tR2_final: out std_logic_vector(7 downto 0);
                        datos_final:out std_logic_vector (7 downto 0));
end sistema_procesador_version3_memoria;

architecture Behavioral of sistema_procesador_version3_memoria is
signal wr: std_logic:='1';
signal di,da: std_logic_vector (7 downto 0);
component procesador_version3
PORT(clk,rst: IN std_logic;
    r_w: OUT std_logic;
    dir: OUT std_logic_vector(7 DOWNTO 0);
--Agrego estas tres seales para poder ver los registros en tiempo real
           ta: out std_logic_vector(7 downto 0);
           tR1: out std_logic_vector(7 downto 0);
           tR2: out std_logic_vector(7 downto 0);
    dat: INOUT std_logic_vector(7 DOWNTO 0));
END component;
component memoria_32instrucciones
   Port ( direcciones : in std_logic_vector(7 downto 0);
          datos : out std_logic_vector(7 downto 0);
          wr : in std_logic;
          cs : in std_logic);
END component;

begin
```

u0: memoria_32instrucciones port map(di,da,wr,cs);
u1: procesador_version3 port map (clk,reset,wr,di,ta_final,tR1_final,tR2_final,da);
wr1<=wr;
dir_final<=di;
datos_final<=da;
end Behavioral;

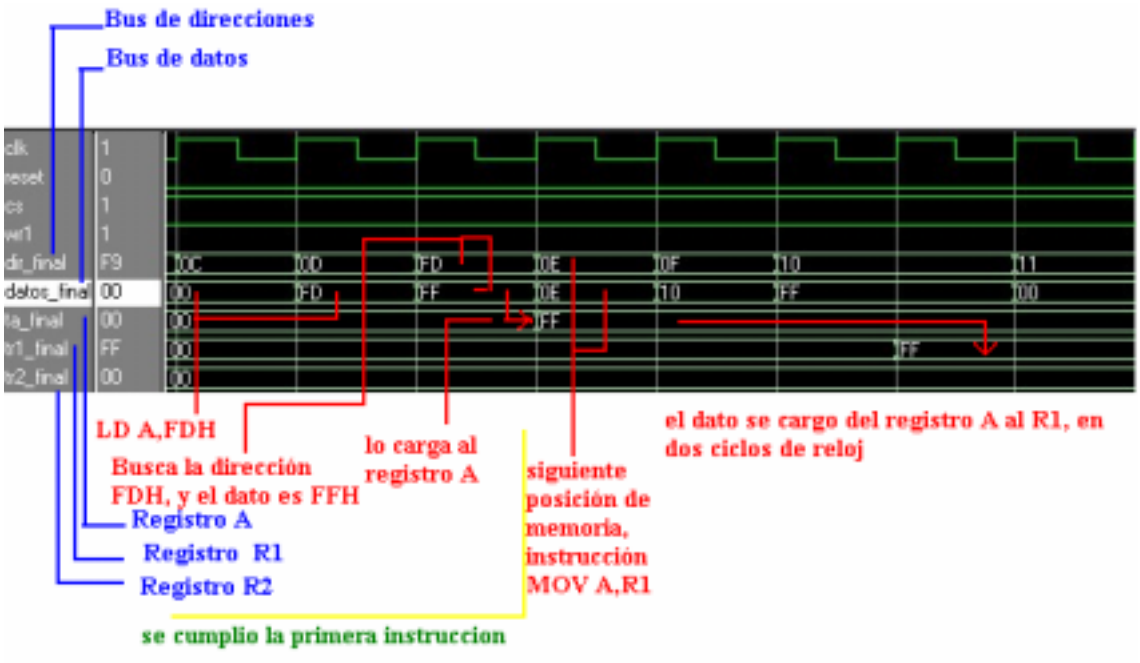
**ESTE ARCHIVO "SISTEMA" NOS SERVIRÁ PARA PROBAR EL MICROPROCESADOR CON 32 INSTRUCCIONES QUE DESARROLLARÉ MAS ADELANTE**

AHORA EL ARCHIVO DE MEMORIA DONDE DEMOSTRAREMOS CON UN MAPEO DIRECTO DE MEMORIA EL FUNCIONAMIENTO DEL MICROPROCESADOR Y DE SUS TRES REGISTROS A, R1, R2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity memoria_32instrucciones is
    Port ( direcciones : in std_logic_vector(7 downto 0);
           datos : out std_logic_vector(7 downto 0);
           wr : in std_logic;
           cs : in std_logic);
end memoria_32instrucciones;
architecture Behavioral of memoria_32instrucciones is

begin
parte1:    process(direcciones,wr,cs)
                begin
                if cs='1' then
                     if wr='1' then
                          case direcciones is
                          when "00000000" => datos <= "00000000";
                          when "00000001" => datos <= "00000001";
                          when "00000010" => datos <= "00000010";
                          when "00000011" => datos <= "00000011";
                          when "00000100" => datos <= "00000100";
                          when "00000101" => datos <= "00000101";
                          when "00000110" => datos <= "00000110";
                          when "00000111" => datos <= "00000111";
                          when "00001000" => datos <= "00001000";
                          when "00001001" => datos <= "00001001";
                          when "00001010" => datos <= "00001010";
                          when "00001011" => datos <= "00001011";
                          when "00001100" => datos <= "00001100";
                          when "00001101" => datos <= "00001101";
                          when "00001110" => datos <= "00001110";
                          when "00001111" => datos <= "00001111";
                          when "00010000" => datos <= "00010000";
                          when "00010001" => datos <= "00010001";
                          when "00011010" => datos <= "00011010";
                          when "00011011" => datos <= "00011011";
```

```
                                when "00011100" => datos <= "00011100";
                                when "00011101" => datos <= "00011101";
                                when "00011110" => datos <= "00011110";
                                when "00011111" => datos <= "00011111";
                                when others => datos <="00000000";
                                end case;
                           end if;
                end if;
                end process;
end Behavioral;
```

OBTENIENDO LA SIGUIENTE GRAFICA



6.- AHORA PARA APROVECHAR MEJOR EL BYTE DE INSTRUCCIÓN, O SEA SUS 5 BITS, VAMOS A AUMENTAR ALGUNAS INSTRUCCIONES MAS, OBTENIENDO ASI 32 INSTRUCCIONES

| HEXADECIMAL | INSTRUCCIÓN | BINARIO |
|---|---|---|
| 00H | LD A,xx | 00000000 |
| 01H | LD xx,A | 00000001 |
| 02H | AND A,xx | 00000010 |
| 03H | ADD A,xx | 00000011 |
| 04H | SUB A,xx | 00000100 |
| 05H | JZ xx | 00000101 |
| 06H | JMP xx | 00000110 |
| 07H | NOP | 00000111 |
| 08H | OR A,xx | 00001000 |
| 09H | XOR A,xx | 00001001 |
| 0AH | NOT A | 00001010 |
| 0BH | INC A | 00001011 |
| 0CH | DEC A | 00001100 |
| 0DH | MOV A,R1 | 00001101 |
| 0EH | MOV R1,A | 00001110 |
| 0FH | MOV A,R2 | 00001111 |
| 10H | MOV R2,A | 00010000 |
| 11H | ROL A | 00010001 |
| 12H | ROL R1 | 00010010 |
| 13H | ROL R2 | 00010011 |
| 14H | ROR A | 00010100 |
| 15H | ROR R1 | 00010101 |
| 16H | ROR R2 | 00010110 |
| 17H | INC R1 | 00010111 |
| 18H | INC R2 | 00011000 |
| 19H | AND A,R1 | 00011001 |
| 1AH | AND A,R2 | 00011010 |
| 1BH | OR A,R1 | 00011011 |
| 1CH | OR A,R2 | 00011100 |
| 1DH | SUM R1,R2 | 00011101 |
| 1EH | SUB R1,R2 | 00011110 |
| 1FH | SUB R2,R1 | 00011111 |

Creamos el archivo para generar este procesador

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY procesador_version3 IS
PORT(clk,rst: IN std_logic;
     r_w: OUT std_logic;
     dir: OUT std_logic_vector(7 DOWNTO 0);
--Agrego estas tres seales para poder ver los registros en
tiempo real
       ta: out std_logic_vector(7 downto 0);
```

```
       tR1: out std_logic_vector(7 downto 0);
       tR2: out std_logic_vector(7 downto 0);
     dat: INOUT std_logic_vector(7 DOWNTO 0));
END procesador_version3;

ARCHITECTURE descripcion OF procesador_version3 IS
  TYPE estado IS
(inicial,busqueda,ejec,ldxxa,ldaxx,anda,adda,suba,oraxx,xoraxx);
  SIGNAL a,pc,carga_a,carga_pc: std_logic_vector(7 DOWNTO 0);
  SIGNAL R1,R2,CARGA_R1,CARGA_R2: std_logic_vector (7 downto 0);
  SIGNAL rdat_in,dat_in,dat_out: std_logic_vector(7 DOWNTO 0);
  SIGNAL rwaux: std_logic;
  SIGNAL presente: estado:=inicial;
--inicio de la funcion +
  FUNCTION "+"(d1,d2: std_logic_vector(7 DOWNTO 0)) RETURN
    std_logic_vector IS
  VARIABLE sal: std_logic_vector(7 DOWNTO 0);
  VARIABLE carry: std_logic:='0';
  VARIABLE ind: INTEGER;
  BEGIN
     FOR ind IN 0 TO 7 LOOP
         sal(ind):=d1(ind) XOR d2(ind) XOR carry;
         carry:=(d1(ind) AND d2(ind)) OR (d1(ind) AND carry) OR
(d2(ind) AND carry);
     END LOOP;
     RETURN sal;
  END "+";
--fin de la funcion +

--inicio de la funcion -
      FUNCTION "-"(d1,d2: std_logic_vector(7 DOWNTO 0)) RETURN
    std_logic_vector IS
  BEGIN
     RETURN d1+((NOT d2)+"00000001");
  END "-";
--fin de la funcion -

BEGIN

fsm:
PROCESS(clk,rst)
BEGIN
 IF rst='1' THEN
    presente<=inicial;
 ELSIF clk='1' AND clk'event THEN
  CASE presente IS
  WHEN inicial =>
     presente<=busqueda;
  WHEN busqueda=>
     IF dat_in(4 DOWNTO 0)="00111" THEN presente<=busqueda;
     ELSE presente<=ejec;
     END IF;
  WHEN ejec =>
     CASE rdat_in(4 DOWNTO 0) IS
     WHEN "00000" =>
        presente<=ldaxx;
```

```
      WHEN "00001" =>
         presente<=ldxxa;
      WHEN "00010" =>
         presente<=anda;
      WHEN "00011" =>
         presente<=adda;
      WHEN "00100" =>
         presente<=suba;
      WHEN "01000" =>
         presente<=oraxx;
      WHEN "01001" =>
         presente<=xoraxx;

      WHEN OTHERS =>      -- jz,jmp
         presente<=busqueda;
      END CASE;
  WHEN OTHERS=>
     presente<=busqueda;
  END CASE;
 END IF;
END PROCESS fsm;

salida:
PROCESS(presente,pc,a,rdat_in,dat_in)
BEGIN
  CASE presente IS
  WHEN inicial =>
    carga_pc<="00000000";     -- PC a 0
    carga_a<="00000000";      -- Acumulador a 0
    carga_R1<="00000000";      -- Acumulador R1 0
    carga_R2<="00000000";      -- Acumulador R2 0
    dir<=pc;                  -- Dirección a 0;
    rwaux<='1';               -- Lectura
  WHEN busqueda=>
    carga_pc<=pc+"00000001";  -- Incrementa PC
    carga_a<=a;               -- Deja A como estaba
    carga_R1<=R1;               -- Deja R1omo estaba
    carga_R2<=R2;               -- Deja R2omo estaba
    dir<=pc;
    rwaux<='1';
  WHEN ejec =>
    CASE rdat_in(4 DOWNTO 0) IS
    WHEN "00101" =>  -- jz
       dir<=pc;
       IF a="00000000" THEN
         carga_pc<=dat_in;   -- Salta si A=0
       END IF;
       rwaux<='1';
       carga_a<=a;
    WHEN "00110" =>  --jmp
       dir<=pc;
       carga_pc<=dat_in;     -- Salta siempre
       rwaux<='1';
       carga_a<=a;
     WHEN "01010" =>
           dir<=pc;
```

```
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_a<=not a;
      WHEN "01011" =>
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_a<=a+"00000001";
      WHEN "01100" =>
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_a<=a-"00000001";
      WHEN "01101" =>
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_a<=R1;
      WHEN "01110" =>
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_R1<=a;
      WHEN "01111" =>
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_a<=R2;
      WHEN "10000" =>
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_R2<=a;
      WHEN "10001" => -- ROL A (rota un bit a la izquierda)
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_a<=a(6)&a(5)&a(4)&a(3)&a(2)&a(1)&a(0)&a(7);
      WHEN "10010" => -- ROL R1 (rota un bit a la izquierda)
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";

carga_R1<=R1(6)&R1(5)&R1(4)&R1(3)&R1(2)&R1(1)&R1(0)&R1(7);
      WHEN "10011" => -- ROL R2 (rota un bit a la izquierda)
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";

carga_R2<=R2(6)&R2(5)&R2(4)&R2(3)&R2(2)&R2(1)&R2(0)&R2(7);
      WHEN "10100" => -- ROR A (rota un bit a la derecha)
             dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
             carga_a<=a(0)&a(7)&a(6)&a(5)&a(4)&a(3)&a(2)&a(1);
      WHEN "10101" => -- ROR R1 (rota un bit a la derecha)
```

```
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";

carga_R1<=R1(0)&R1(7)&R1(6)&R1(5)&R1(4)&R1(3)&R1(2)&R1(1);
      WHEN "10110" => -- ROR R2 (rota un bit a la derecha)
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";

carga_R2<=R2(0)&R2(7)&R2(6)&R2(5)&R2(4)&R2(3)&R2(2)&R2(1);
      WHEN "10111" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_R1<=R1+"00000001";
      WHEN "11000" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_R2<=R2+"00000001";
      WHEN "11001" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_a<=a and R1;
      WHEN "11010" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_a<=a and R2;
      WHEN "11011" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_a<=a or R1;
      WHEN "11100" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_a<=a or R2;
      WHEN "11101" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_a<=R1+R2;
      WHEN "11110" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_a<=R1-R2;
      WHEN "11111" =>
            dir<=pc;
        rwaux<='1';
        carga_pc<=pc+"00000001";
            carga_a<=R2-R1;
```

```
    WHEN OTHERS =>   -- Por defecto para el resto:
       rwaux<='1';   -- Leer
       dir<=pc;      -- Dirección del operando
       carga_pc<=pc;
       carga_a<=a;
            carga_R1<=R1;
            carga_R2<=R2;
    END CASE;
  WHEN ldaxx =>
    rwaux<='1';
    carga_a<=dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN ldxxa =>
    rwaux<='0';   -- Único que escribe
    carga_a<=a;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN anda =>
    rwaux<='1';
    carga_a<=a AND dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN adda =>
    rwaux<='1';
    carga_a<=a+dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN suba =>
    rwaux<='1';
    carga_a<=a-dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN oraxx =>
    rwaux<='1';
    carga_a<=a or dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  WHEN xoraxx =>
    rwaux<='1';
    carga_a<=a xor dat_in;
    carga_pc<=pc+"00000001";
    dir<=rdat_in;
  END CASE;
END PROCESS salida;

registro_de_entrada:
PROCESS(clk,rst)
BEGIN
  IF rst='1' THEN rdat_in<="00000000";
  ELSIF clk='1' AND clk'event THEN rdat_in<=dat_in;
  END IF;
END PROCESS registro_de_entrada;

regs_con_carga:
```

```
PROCESS(clk,rst)
BEGIN
  IF rst='1' THEN
    a <="00000000";
      R1<="00000000";
      R2<="00000000";
    pc<="00000000";
  ELSIF clk='1' AND clk'event THEN
    a <=carga_a;
      R1<=carga_R1;
      R2<=carga_R2;
    pc<=carga_pc;
  END IF;
END PROCESS regs_con_carga;

r_w<=rwaux;   -- Para leer la salida.

-- Triestado de salida:
dat_in<=dat;
dat_out<=a;
dat<=dat_out WHEN rwaux='0' ELSE "ZZZZZZZZ";
--Zoma aumentada para pruebas
ta<=a;
tR1<=R1;
tR2<=R2;

END descripcion;
```

CON EL SIGUIENTE SISTEMA, Y CON LAS ARCHIVOS DE MEMORIA Y SIMULACIÓN ANTERIORES

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sistema_procesador_version3_memoria is
    Port ( clk : in std_logic;
           reset : in std_logic;
           cs : in std_logic;
                  wr1: out std_logic;
                  dir_final:out std_logic_vector (7 downto 0);
             ta_final: out std_logic_vector(7 downto 0);
                  tR1_final: out std_logic_vector(7 downto 0);
                  tR2_final: out std_logic_vector(7 downto 0);
                  datos_final:out std_logic_vector (7 downto
0));
end sistema_procesador_version3_memoria;

architecture Behavioral of sistema_procesador_version3_memoria
is
signal wr: std_logic:='1';
signal di,da: std_logic_vector (7 downto 0);
component procesador_version3
PORT(clk,rst: IN std_logic;
```

```
     r_w: OUT std_logic;
     dir: OUT std_logic_vector(7 DOWNTO 0);
--Agrego estas tres seales para poder ver los registros en
tiempo real
       ta: out std_logic_vector(7 downto 0);
       tR1: out std_logic_vector(7 downto 0);
       tR2: out std_logic_vector(7 downto 0);
     dat: INOUT std_logic_vector(7 DOWNTO 0));
END component;
component memoria_32instrucciones
    Port ( direcciones : in std_logic_vector(7 downto 0);
           datos : out std_logic_vector(7 downto 0);
           wr : in std_logic;
           cs : in std_logic);
END component;

begin

u0: memoria_32instrucciones port map(di,da,wr,cs);
u1: procesador_version3 port map
(clk,reset,wr,di,ta_final,tR1_final,tR2_final,da);
wr1<=wr;
dir_final<=di;
datos_final<=da;
end Behavioral;
```

SE OBTIENE LAS SIGUIENTES GRÁFICAS